Geometric Operators on Boolean Functions

Frisvad, Jeppe Revall; Falster, Peter

Publication date:
2007

Document Version
Publisher's PDF, also known as Version of record

Geometric Operators on Boolean Functions

Jeppe Revall Frisvad[*] and **Peter Falster**[**]

Informatics and Mathematical Modelling, Technical University of Denmark

Key words array-based logic, Boolean functions, geometric operators, inference, propositional reasoning.

Abstract
In truth-functional propositional logic, any propositional formula represents a Boolean function (according to some valuation of the formula). We describe operators based on Decartes' concept of constructing coordinate systems, for translation of a propositional formula to the image of a Boolean function. With this image of a Boolean function corresponding to a propositional formula, we prove that the orthogonal projection operator leads to a theorem describing all rules of inference in propositional reasoning. In other words, we can capture all kinds of inference in propositional logic by means of a few geometric operators working on the images of Boolean functions. The operators we describe, arise from the niche area of array-based logic and have previously been tightly bound to an array-based representation of Boolean functions. We redefine the operators in an abstract form to make them independent of representation such that we no longer need to be much concerned with the form of the Boolean functions. Knowing that the operators can easily be implemented (as they have been in array-based logic), shows the advantage they give with respect to automated reasoning.

1 Introduction

If we accept a truth-functional conception of propositional logic, any propositional formula represents a Boolean function. Taking this point of view, we describe using operators on Boolean functions: (a) An efficient mechanical way for translation of propositional formulae to Boolean functions (Sections 2.2-2.5); (b) how any kind of inference in propositional logic can be captured by geometrical concepts (Sections 2.6-2.8); (c) that regardless of the representation employed for the Boolean functions, the operators are applicable (Section 3).

An analogy of Boolean logic with coordinate-geometry was shown by Mautner in 1946 [17]. He introduced the idea of a many-dimensional *logical coordinate system*, i.e. a discrete cartesian coordinate system where each axis represents a Boolean variable, and thereby he connected Boolean logic to the mathematical group of geometric transformations. Mautner's investigations go far beyond this into the realm of invariant theory and Boolean tensor algebra. These algebraic investigations are not necessary for the theory we are about to develop. Realizing that we can treat the image of a Boolean function as geometry is, however, all-important for appreciation of the following sections. At some points in Section 2 the reader may find that our theory could advantageously be reformulated using Boolean tensors instead of Boolean functions. We are aware of this, but have for the time being chosen not to elongate this paper by the algebra necessary for a reformulation.

In the spirit of Mautner's analogy, the foundation of the geometric operators which we give a more abstract form in this paper, were laid by Franksen in 1979 [7]. He showed that disjunctive projection in a logical coordinate system can "prove the theorems of divalent logic by computation" (projection is described in Sec. 2.6); that an outer product can construct the relation between two variables on matrix form; and that "the operation of putting indices equal, is the operational implementation of repeated propositions in a propositional function" (i.e. colligation, see Sec. 2.5).

Through a generalization of these fundamental operations, enabling them to operate on many-dimensional arrays, the niche area of array-based logic was developed to its present state in [8, 9, 10, 18, 23]. The functional or operational notation described in *array theory* has traditionally been used to account for the operators used in array-based logic. Array theory was developed by Trenchard More, see eg. [19, 20].

[*] e-mail: jrf@imm.dtu.dk
[**] e-mail: pfa@imm.dtu.dk

The purpose of this paper is to propose a synergy of the abstract, representation independent notation used for Boolean functions (e.g. in [28]) and the array theoretic notation used in array-based logic. Through such a synergy, we will be able to show that geometric operations on the images of Boolean functions make sense at a high level of abstraction in propositional reasoning. In particular, we are able to show that the image of a Boolean function represented by an arbitrary propositional rule set, can be found by use of outer products and the picking of diagonal hyperplanes (i.e. the operation of setting indices equal or colligation) instead of finding the value of the rule set for every possible valuation of the propositional variables which it contains. In addition, we are able not only to prove the theorems of divalent logic by computation as did Franksen, but also to prove that disjunctive projection leads to a formula which capture all rules of inference in propositional reasoning (see Sec. 2.7).

2 Boolean Functions

Take an arbitrary rule set (or set of propositional formulae) describing the relation between n propositional variables $p_1, \ldots, p_n$. By a valuation $v : \{p_1, \ldots, p_n\} \to \{0, 1\}$, any such rule set represents a function f taking n Boolean values as argument and returning a single Boolean value. Each argument of f corresponds to one of the propositions in the rule set, and the returned value $f(a_1, \ldots, a_n) \in \{0, 1\}$ for $a_k = v(p_k)$ with $k = 1, \ldots, n$, states whether a particular valuation (or interpretation) of the propositional variables is true (1) or false (0).

It will often be the case that only some of the propositional variables are asserted (or known, or bound) to be true or false. Suppose we know the value of p_i and p_j, then a new Boolean function f_d is desired such that it is represented by a propositional formula A in which only the variables $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{j-1}, p_{j+1}, \ldots, p_n$ appear. The derived function f_d should be found according to the known values of p_i and p_j. This is accomplished simply by letting f_d equal f with fixed values $a_i = v(p_i)$ and $a_j = v(p_j)$. In that way, the Boolean function f_d corresponds to the result of deductive inference on these fixed values. This is the kind of inference described by the Stoic modi (modus ponens, modus tollens, etc.) and we say that it relies on external influences, that is, the fixation of some propositional variables leading to a conclusion according to a rule set.

The rule set itself can also lead to a conclusion on the relation between a subset of the propositional variables. We could say that such conclusions are internally present in the rule set. This kind of inference is inherent in the syllogistic reasoning founded in the Greek school of logic, especially in Aristotle's Prior and Posterior Analytics. While the logic of Aristotle is often thought of as notions for predicate logic, it should be realized that any formula of predicate calculus over a finite domain, can be translated to a formula of propositional calculus. When we refer to the Aristotelian syllogism in propositional calculus, we refer to the transitive law

$$\frac{p_1 \Rightarrow p_2 \quad , \quad p_2 \Rightarrow p_3}{p_1 \Rightarrow p_3}$$

which is not a direct translation of the syllogism in predicate calculus, since that would require us to know the domain of the predicates involved. The transitive law is rather an analogy.

To find a conclusion "internally present in a rule set", we seek a function f'_d describing the relation between only some of the propositional variables appearing in the rule set representing f_d, but in this case none of the variables are asserted. In the traditional Aristotelian syllogism one intermediate variable is eliminated. Conceptually the idea is to find the relation between a subset of variables appearing in the rule set. Suppose we wish to exclude not one, but two propositional variables from a set of formulae, say p_i and p_j. This is done by the principle of excluded middle ($p \vee \neg p$) such that f'_d returns 1 if any one of the combinations $(a_i, a_j) \in \{0, 1\}^2$ returns 1 when a_i and a_j are inserted as arguments of f. In other words,

$$\begin{aligned} &f'_d(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n) \\ &\quad = \bigvee_{b_1, b_2 \in \{0,1\}} f(a_1, \ldots, a_{i-1}, b_1, a_{i+1}, \ldots, a_{j-1}, b_2, a_{j+1}, \ldots, a_n) \ , \end{aligned}$$

where $i, j, n \in \mathbb{N}$ and $i < j < n$.

These two concepts of inference are surprisingly general. In fact, most forms of inference can be based on the elimination of variables as it is described above. In the following, we define a number of operators some of which have a geometrical meaning in a logical coordinate system. These operators are inspired by the two concepts of inference and can not only prove, but also replace the logical rules of inference (the Stoic modi, the Aristotelian syllogism, etc.).

Table 1 The four possible Boolean functions $f_i \in B_1$, $i = 1, 2, 3, 4$. The image is ordered such that the first number in the table is $f_i(0)$ and the second is $f_i(1)$.

Image	State	Term
0 1	True	Affirmation
1 0	False	Negation
1 1	Indefinite	Tautology
0 0	Impossible	Contradiction

2.1 Fundamentals

Definition 2.1.1 (Boolean functions) *Let $B_{n,m}$, where $n, m \in \mathbb{N}$, denote the set of* Boolean functions $f : \{0,1\}^n \to \{0,1\}^m$*, and let B_n stand for $B_{n,1}$.*

For functions $f \in B_n$ there are 2^n different inputs each of which can be mapped to 0 or 1, hence, there exist 2^{2^n} functions in B_n [26]. Consider the $2^{2^1} = 4$ possible Boolean functions in B_1, see Table 1. A function $f \in B_1$ can specify the state of a single logical proposition.

Definition 2.1.2 *For every ordered pair of Boolean values $(a, b) \in \{0,1\}^2$ there exists exactly one Boolean function $f_{a,b} \in B_1$ such that $f_{a,b}(0) = a$ and $f_{a,b}(1) = b$.*

Applying a fundamental principle of mathematics, namely the splitting of arguments, any Boolean function $f : \{0,1\}^{k+n} \to \{0,1\}$ can also be described as a function

$$f : \{0,1\}^k \times \{0,1\}^n \to \{0,1\} = \{0,1\}^k \to (\{0,1\}^n \to \{0,1\}) = \{0,1\}^k \to B_n$$

from $\{0,1\}^k$ into B_n. The splitting of arguments of a Boolean function is essential to our theory. Therefore we introduce

Definition 2.1.3 (Nested Boolean functions) *Let B_n^k denote a set of* nested Boolean functions $f : \{0,1\}^k \to B_n$*, such that $(a_1, \ldots, a_k) \in \{0,1\}^k$ maps to a Boolean function $g \in B_n$.*

Strictly speaking the notion of a nested Boolean function is not necessary since $B_n^k = B_{n+k}$, but it will ease the introduction of the operators presented in the following.

Proposition 2.1.4 *A nested Boolean function $f \in B_n^k$ contains* 2^k nested elements $g_i \in B_n$*, $i = 1, \ldots, 2^k$.*

A note on notation. Throughout the paper we employ left associativity with respect to operators and arguments as well as occasional infix notation. We employ the notation that for any $g \in B_k$, any $f_1, \ldots, f_k \in B_n$, and any $\vec{a} \in \{0,1\}^n$, where $n, k \in \mathbb{N}$,

$$g(f_1(\vec{a}), \ldots, f_k(\vec{a})) = g(f_1, \ldots, f_k)(\vec{a})$$

and we allow for infix notation if $k = 2$. In addition, we employ the common notation for indexing. Let $I = \{1, \ldots, n\}$ be an index set and let $n, m \in \mathbb{N}$ be natural numbers. Given an $\vec{x} = (x_1, \ldots, x_n)$ and an index vector $\vec{\imath} \in I^m$, we have $\vec{x}_{\vec{\imath}} = (x_{i_1}, \ldots, x_{i_m})$.

In the following section a propositional rule set is given a formal definition.

2.2 Rule Sets

In reasoning, an autonomous agent, or whatever system considered, is equipped with a set of propositional formulae often referred to as a rule base or a rule set. The propositional formulae can be established by any choice of connectives which correspond to Boolean functions, e.g. $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$. The connectives $\neg$ (negation) and $\vee$ (disjunction) suffice to construct a formula representing any possible Boolean function [26], but for the theory we are about to develop, the choice of connectives is of no consequence. Boolean functions corresponding to commonly employed connectives will be referred to by the names listed in Table 2.

In the remainder of the text, we let $p_1, p_2, \ldots$ refer to propositional variables. A rule base or rule set for reasoning is then defined by

Table 2 The names we use for Boolean functions corresponding to commonly employed connectives.

Connective	Corresponding Boolean function	Term
	aff	Affirmation
$\neg$	non	Negation
$\wedge$	et	Conjunction
$\vee$	vel	Disjunction
$\Rightarrow$	imp	Implication
$\Leftrightarrow$	bii	Biimplication
$\Leftarrow$	$cimp$	Converse implication

Definition 2.2.1 (Propositional rule set) *A* propositional rule set *is a propositional formula* $A = A(p_1, \ldots, p_n)$ *in which a finite number of propositional variables appear. A propositional rule set is constructed from a set of propositional formulae* R *by*

$$A = \bigwedge_{C \in R} C \ .$$

We say that the rule set is fulfilled *by a valuation (or interpretation)* $v : \{p_1, \ldots, p_n\} \to \{0, 1\}$ *if* $v(A) = 1$*, where* $v(A)$ *denotes the value of* A *by the valuation* v*.*

The purpose of the following three sections is to construct a tool which can translate a propositional rule set $A = A(p_1, \ldots, p_n)$ to the corresponding Boolean function $f_A : \{0, 1\}^n \to \{0, 1\}$ defined by

$$f_A(a_1, \ldots, a_n) = v(A(p_1, \ldots, p_n)) \quad , \quad \text{where } a_i = v(p_i) \text{ for } i = 1, \ldots, n \ .$$

While this may seem like a superfluous thing to do, it is necessary since we intend to do reasoning by projections in the image of the Boolean function corresponding to a rule set. By definition the image of a Boolean function is easily, but very inefficiently, determined through computation of every possible valuation of the formula representing the Boolean function. However, there is a better alternative which is based on the construction of coordinate-systems. That is what we wish to advocate in Sections 2.3, 2.4, and 2.5.

2.3 Reduction

Any pair of Boolean values $(a, b) \in \{0, 1\}^2$ corresponds to an unary Boolean function $f_{a,b} \in B_1$ (Def. 2.1.2) and, of course, any binary Boolean function $g \in B_2$ can be invoked on a pair of Boolean values to produce a single Boolean value $c = g(a, b)$. We find it convenient to introduce an operator named REDUCE which allows us to invoke a binary Boolean function on an unary Boolean function such that

$$c = g(a, b) = \mathrm{REDUCE}(g)(f_{a,b}) \ .$$

Sometimes it is also sensible to reduce an arbitrary Boolean function according to one of the binary Boolean functions (in particular this makes sense for disjunction and conjunction, that is, for vel and et respectively, cf. Tab. 2). Therefore we may as well give REDUCE a more general definition. By T_n for $n \in \mathbb{N}$ we denote the set of all functionals $\chi_n : B_n \to \{0, 1\}$, then we have

Definition 2.3.1 (REDUCE) *Let* $\chi_n^f \in T_n$ *for* $f \in B_2$ *be the functional defined for* $g \in B_n$ *by*

$$\chi_n^f = f(f(\ldots(f(g(\vec{a}_1), g(\vec{a}_2)), g(\vec{a}_3)), \ldots), g(\vec{a}_{2^n})) \ ,$$

where $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{2^n}$ *denotes the canonical enumeration of the* n*-dimensional Boolean vectors* $\vec{a}_i \in \{0, 1\}^n$ *given by* $\vec{a}_i = (i_1, \ldots, i_n)$ *with* $i_v = \lfloor (i-1)/2^{v-1} \rfloor \mod 2$ *for* $v = 1, \ldots, n$*.*
Then the operator $\mathrm{REDUCE} : B_2 \to T_n$ *is defined by*

$$\mathrm{REDUCE}_n(f) = \chi_n^f \ .$$

The idea of this operator is to allow for reduction of a nested Boolean function $f_{n,k} \in B_n^k$ to a Boolean function $g_k \in B_k$ using one of the binary Boolean functions. To accomplish this, we define an operator reducing *each* possible nested element of a nested Boolean function to a single Boolean value. By $U_{n,k}$ we denote the set of all operators $\psi_{n,k} : B_n^k \to B_k$, then we have

Definition 2.3.2 (EACH) *The operator* $\mathrm{EACH}_{n,k} : T_n \to U_{n,k}$ *is defined for* $\chi_n \in T_n$ *and* $f_{n,k} \in B_{n,k}$ *by*

$$\mathrm{EACH}_{n,k}(\chi_n)(f_{n,k}) = \chi_n \circ f_{n,k} \ ,$$

where $\circ$ *is the composition operator, i.e.*

$$\mathrm{EACH}_{n,k}(\chi_n)(f_{n,k})(a_1, \ldots, a_k) = \chi_n(f_{n,k}(a_1, \ldots, a_k))$$

for all $a_1, \ldots, a_k \in \{0, 1\}$.

Now, pick an arbitrary nested Boolean function $f_{n,k} \in B_n^k$. Using the binary Boolean function $vel \in B_2$ (i.e. disjunction), $f_{n,k}$ can be reduced to a Boolean function $g_k \in B_k$ in the following way:

$$g_k = \mathrm{EACH}_{n,k}(\mathrm{REDUCE}_n(vel))(f_{n,k}) \ .$$

This is referred to as a disjunctive reduction of a nested Boolean function. If, for example, $f_{n,k}$ is defined by

$$f_{n,k}(y_1, \ldots, y_n)(x_1, \ldots, x_k) = f_{n+k}(y_1, \ldots, y_n, x_1, \ldots, x_k) \ ,$$

then

$$\mathrm{EACH}_{n,k}(\mathrm{REDUCE}_n(vel))(f_{n,k})(x_1, \ldots, x_k) = \bigvee_{(a_1,\ldots,a_n)\in\{0,1\}} f_{n+k}(a_1, \ldots, a_n, x_1, \ldots, x_k) \ .$$

For any $n \in \mathbb{N}$ we refer to the functional $\mathrm{REDUCE}_n(f_2)$ as the *reduction transform of* $f_2 \in B_2$, and to shorten the notation, we denote $\mathrm{REDUCE}_n(f_2)$ by the same symbol as the connective corresponding to f_2. This means that a disjunctive reduction of the Boolean function $f_{n,k}$ can be written as

$$g_k = \mathrm{EACH}_{n,k}(\vee)(f_{n,k})$$

and that, for example,

$$\Rightarrow(\mathit{aff}) = imp(0, 1) = 1 \ .$$

2.4 Cartesian Product and Outer Product

To compute the image of a Boolean function from the formula representing it in propositional calculus without testing each possible valuation, we must find a way to rewrite the formula into the world of Boolean functions. As the first step in this endeavor, we replace each occurrence of a propositional variable in the formula by a Boolean function in B_1. This is done in a manner such that any non-negated variable (e.g. p) is replaced by *aff* and any negated variable (e.g. $\neg q$) is replaced by non. These Boolean functions replacing propositional variables are then connected by application of an operator to the connectives in the formula (or, more precisely, to the reduction transform of the binary Boolean function corresponding to the connectives in the formula). This operator is founded in the concept of a cartesian product between the images of Boolean functions. We have

Definition 2.4.1 (cart) *Let* $\mathrm{cart}_{n,m} : B_n \times B_m \to B_1^{n+m}$, *where* $n, m \in \mathbb{N}$, *denote the cartesian product of two Boolean functions, such that if* $f_n \in B_n$ *and* $g_m \in B_m$*:*

$$\begin{aligned} \mathrm{cart}_{n,m}(f_n, g_m)(x_1, \ldots, x_{n+m})(0) &= f_n(x_1, \ldots, x_n) \\ \mathrm{cart}_{n,m}(f_n, g_m)(x_1, \ldots, x_{n+m})(1) &= g_m(x_{n+1}, \ldots, x_{n+m}) \ , \end{aligned}$$

where $x_1, \ldots, x_{n+m}$ *are Boolean variables.*

Note that cart finds unary Boolean functions corresponding to a pair of function values. This is closely related to the usual notion of a cartesian product, only cart does not find the cartesian product of two sets, but rather the cartesian product of two function images.

The nested Boolean function resulting from the cartesian product of two Boolean functions has exactly a form which can be reduced to a normal Boolean function by the operator EACH and any of the reduction transforms corresponding to a connective (cf. Sec. 2.3). Using cart and EACH we can define the outer product between two arbitrary Boolean functions according to the reduction transform of a binary Boolean function. This outer product is the operator we use to translate the connectives in a formula of propositional calculus into the world of Boolean functions. As in the previous section let T_n for $n \in \mathbb{N}$ denote the set of all functionals $\chi_n : B_n \to \{0,1\}$ and let $V_{n,m}$ for all $n, m \in \mathbb{N}$ denote the set of all operators $\zeta_{n,m} : B_n \times B_m \to B_{n+m}$.

Definition 2.4.2 (OUTER) *The outer product operator* $\mathrm{OUTER}_{n,m} : T_1 \to V_{n,m}$ *is defined for* $\chi_1 \in T_1$, $f_n \in B_n$, *and* $g_m \in B_m$ *by*

$$\mathrm{OUTER}_{n,m}(\chi_1)(f_n, g_m) = \mathrm{EACH}_{1,n+m}(\chi_1)(\mathrm{cart}_{n,m}(f_n, g_m)) \ .$$

Infix notation is allowed for use with OUTER *such that*

$$\mathrm{OUTER}_{n,m}(\chi_1)(f_n, g_m) = f_n \,\mathrm{OUTER}_{n,m}(\chi_1)\ g_m \ .$$

In the preceding paragraphs we have described replacement of propositional variables with unary Boolean functions and replacement of connectives with OUTER applied to the reduction transforms corresponding to them. While this may be a way to translate a propositional formula A to the Boolean function f_A which it represents, it remains to be shown that the resulting Boolean function actually has the image that we desire. In the following, let $\mathrm{op}_1, \mathrm{op}_2, \ldots$ denote binary connectives.

Theorem 2.4.3 *Let* $A = A(p_1, \ldots, p_n)$ *denote the propositional formula given by*

$$A_1 \ \mathrm{op}_1 \ A_2 \ \mathrm{op}_2 \ \ldots \ \mathrm{op}_{n-1} \ A_n \ ,$$

where A_i, $i = 1, \ldots, n$, *are sub-formulae for which the image of the corresponding Boolean function* f_{A_i} *is known. The sub-formulae may contain any of the variables* $p_1, \ldots, p_n$. *Then for the Boolean function* f_A *represented by* A *it holds for any valuation* $v : \{p_1, \ldots, p_n\} \to \{0,1\}$ *that*

$$\begin{aligned} f_A(a_1, \ldots, a_n) &= v(A(p_1, \ldots, p_n)) \\ &= (f_{A_1} \ \mathrm{OUTER}(\mathrm{op}_1) \ \ldots \ \mathrm{OUTER}(\mathrm{op}_{n-1}) \ f_{A_n})(a_1, \ldots, a_n) \ , \end{aligned}$$

where $a_i = v(p_i)$ *for* $i = 1, \ldots, n$.

P r o o f. The result follows immediately from the definitions given prior to the theorem. As we work with left associativity it suffices to prove that

$$f_{A_1 \ \mathrm{op} \ A_2} = f_{A_1} \ \mathrm{OUTER}(\mathrm{op}) \ f_{A_2} \ ,$$

where $f_{A_1} \in B_n$ and $f_{A_2} \in B_m$ are two arbitrary Boolean functions corresponding to the sub-formulae $A_1(p_1, \ldots, p_n)$ and $A_2(p_{n+1}, \ldots, p_{n+m})$. Let $b \in B_2$ be the binary Boolean function corresponding the connective op. Then

$$\begin{aligned} f_{A_1} \ \mathrm{OUTER}_{n,m}(\mathrm{op}) \ f_{A_2} &= \mathrm{EACH}_{1,n+m}(\mathrm{REDUCE}_1(b))(\mathrm{cart}_{n,m}(f_{A_1}, f_{A_2})) \\ &= \mathrm{REDUCE}_1(b) \circ \mathrm{cart}_{n,m}(f_{A_1}, f_{A_2}) \ . \end{aligned}$$

Continuing with an arbitrary valuation $a_i = v(p_i)$ for $i = 1, \ldots, n+m$ we get

$$\begin{aligned} &(\mathrm{REDUCE}_1(b) \circ \mathrm{cart}_{n,m}(f_{A_1}, f_{A_2}))(a_1, \ldots, a_{n+m}) \\ &\quad = \mathrm{REDUCE}_1(b)(\mathrm{cart}_{n,m}(f_{A_1}, f_{A_2})(a_1, \ldots, a_n + m)) \\ &\quad = b(f_{A_1}(a_1, \ldots, a_n), f_{A_2}(a_{n+1}, \ldots, a_{n+m})) \end{aligned}$$

which is the value of $f_{A_1 \text{ op } A_2}$ for the given valuation. To complete the proof, we must be certain that any formula can be decomposed into sub-formulae with a known image, and binary connectives in-between them. This is certain since we can always reach sub-formulae which are either p_i or $\neg p_i$, $i = 1, \ldots, n+m$, for which the corresponding Boolean functions are *aff* and non respectively. □

As it is the case for the traditional cartesian product between sets, the list of arguments is ordered for the function returned by cart and, hence, it is also ordered for for the function returned by OUTER. This means that after translation of a formula to a Boolean function, the resulting Boolean function will have an argument for each occurrence of each propositional variable in the formula. And the arguments correspond to the valuation of variables in the same order as the variables appear in the formula.

If we had imposed the necessary algebra on our theory and described it using Boolean tensors rather than Boolean functions, then OUTER would denote the traditional outer product between two Boolean tensors (only according to an arbitrary binary Boolean function instead of multiplication). Hence, we have shown that an outer product can be employed for translation of a propositional formula to the Boolean function which it represents. This is most often far more efficient than computation of every possible valuation for the formula. The process we have described in which simple sub-formulae are replaced by known Boolean functions and connected using reduction transforms corresponding to the connectives in-between the sub-formulae, works for any choice of connectives and any representation of the Boolean functions.

Example 2.4.4 Consider the following rule set composed of a set of two propositional formulae:

Rule 1	$p_1 \Rightarrow p_2$
Rule 2	$p_2 \Rightarrow p_3$.

This is equivalent to the formula $A(p_1, p_2, p_3) = (p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_3)$. A Boolean function $f_4 \in B_4$ corresponding to the rule set can be constructed as follows:

$$\begin{aligned} r_1 = r_2 &= \mathit{aff}\ \mathrm{OUTER}_{1,1}(\Rightarrow)\ \mathit{aff} \\ f_4 &= r_1\, \mathrm{OUTER}_{2,2}(\wedge)\, r_2\ , \end{aligned}$$

where $r_1, r_2 \in B_2$. The resulting function f_4 is effectively a Boolean function corresponding to the rule set. If $a_i = v(p_i)$ for $i = 1, 2, 3$, $f_4(a_1, a_2, a_2, a_3)$ returns whether a particular valuation fulfills the rule set or not. This follows from Theorem 2.4.3:

$$\begin{aligned} f_4(a_1, a_2, a_2, a_3) &= \mathrm{OUTER}_{2,2}(\wedge)(r_1, r_2)(a_1, a_2, a_2, a_3) \\ &= et(r_1(a_1, a_2), r_2(a_2, a_3)) \\ &= et\,((\mathit{aff}\ \mathrm{OUTER}_{1,1}(\Rightarrow)\ \mathit{aff})(a_1, a_2), (\mathit{aff}\ \mathrm{OUTER}_{1,1}(\Rightarrow)\ \mathit{aff})(a_2, a_3)) \\ &= et\,(imp(\mathit{aff}\,(a_1), \mathit{aff}\,(a_2)), imp(\mathit{aff}\,(a_2), \mathit{aff}\,(a_3))) \\ &= et(imp(a_1, a_2), imp(a_2, a_3)) \\ &= v((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_3))\ . \end{aligned}$$

Note that $r_1 = r_2 = imp$ and, hence, we could have constructed f_4 merely as

$$f_4 = imp\ \mathrm{OUTER}_{2,2}(\wedge)\ imp\ .$$

2.5 Colligation

It shows in Example 2.4.4 that the function obtained after a concatenation may have several arguments valuated by the same propositional variable. This is clearly inexpedient. It may also be desirable to rearrange the list of arguments. To handle these issues we have

Definition 2.5.1 (fuse) *Let $I = \{1, \ldots, k\}$ be an index set, and let $n, k \in \mathbb{N}$ be natural numbers such that $k < n$, then $\mathrm{fuse}_{n,k} : I^n \times B_n \to B_k$ is defined for $f_n \in B_n$ and $\vec{\imath} \in I^n$ by*

$$\mathrm{fuse}_{n,k}(\vec{\imath}, f_n)(x_1, \ldots, x_k) = f_n(x_{i_1}, \ldots, x_{i_n})$$

where $x_1, \ldots, x_k$ are Boolean variables.

The process of setting two arguments of a Boolean function equal to each other is referred to as *colligation*. A term used by Peirce [24] and Whewell before him. See also the discussion by Franksen and Falster [10]. Geometrically we pick a diagonal hyperplane in the image of a function $f \in B_n$ to obtain the image of a function $g \in B_{n-1}$. fuse does this repeatedly until a function in B_k, $k < n$, is obtained. Hence, fuse is the first of the operators we have described which has a direct geometric interpretation in a logical coordinate-system. The importance of fuse shows through

Proposition 2.5.2 *The image of a Boolean function represented by a propositional rule set, can be found through operations on* the images *of known Boolean functions corresponding to the formulae and sub-formulae composing the rule set.*

The result follows since OUTER and fuse can work exclusively on the images of Boolean functions. Through outer products of image spaces according to reduction transforms corresponding to connectives, OUTER can translate any propositional rule set into a Boolean function with one argument for each appearance of each propositional variable in the rule set (cf. Theorem 2.4.3). Subsequently fuse can eliminate the redundant arguments and rearrange the remaining arguments as appropriate by picking diagonals and interchanging axes in the image of the translated Boolean function. In this way, the *colligated form* of a Boolean function represented by a propositional rule set, can be found. Here colligated form is defined by

Definition 2.5.3 (colligated form) *Let A be the formula describing a propositional rule set and let f_A be the Boolean function represented by the rule set. Then the* colligated form of f_A *is a function equivalent to f_A in which no arguments are valuated by the same propositional variable in A. A function in its colligated form is referred to as a* colligated Boolean function.

Example 2.5.4 In continuance of Example 2.4.4 we can now eliminate the redundant a_2 argument and obtain a function $f_3 \in B_3$ describing the same relation between p_1, p_2, and p_3. This could be written as

$$f_3(a_1, a_2, a_3) = f_4(a_1, a_2, a_2, a_3) \ ,$$

or at a higher level of abstraction as

$$f_3 = \mathrm{fuse}_{4,3}((1, 2, 2, 3), f_4) \ .$$

2.6 Projection

Since the picking of a diagonal hyperplane in the image of a Boolean function is useful, it might be interesting to define and interpret *projection*.

Let $f \in B_n$ be the colligated form of a Boolean function represented by a propositional rule set in which the propositional variables $p_1, \ldots, p_n$ appear. Suppose we want to project the image of f on a subspace spanned by $k < n$ of the Boolean variables which f takes as argument. Thereby we would obtain a function $g \in B_k$ describing, according to the projection, the relation between the remaining Boolean variables valuated by $p_{i_1}, \ldots, p_{i_k}$, where the indices $i_1, \ldots, i_k \in \{1, \ldots, n\}$ are mutually distinct.

To perform such a projection we must first be able to split the image of f. This is exactly the point of nested Boolean functions. We have

Definition 2.6.1 (split) *Let $I = \{1, \ldots, n\}$ be an index set, and let $n, k \in \mathbb{N}$ be natural numbers such that $k < n$, then $\mathrm{split}_{n,k} : I^{n-k} \times B_n \to B_{n-k}^k$ is defined for $f_n \in B_n$ and $\vec{\imath} \in I^{n-k}$ by*

$$\mathrm{split}_{n,k}(\vec{\imath}, f_n)(x_{j_1}, \ldots, x_{j_k})(x_{i_1}, \ldots, x_{i_{n-k}}) = f_n(x_1, \ldots, x_n) \ ,$$

where $x_1, \ldots, x_n$ are Boolean variables. If $k > 0$, then $\vec{\jmath} \in I^k$ exists. Otherwise $\mathrm{split}(\vec{\imath}, f_n)() = f_n$. All indices in $\vec{\imath}$ and $\vec{\jmath}$ must be mutually distinct. Furthermore the indices in $\vec{\jmath}$ are ordered such that $j_v < j_{v+1}$ for $v = 1, \ldots, k-1$. Thus $\vec{\jmath}$ is given implicitly by the indices in I which are not in $\vec{\imath}$.

Thinking of a circle (both circumference and interior of the circle) describing the projection of a sphere on a plane, we may similarly project a relation (such as f mentioned before) on a subspace of its image. This is done through a disjunctive reduction of the nested Boolean function found using split. As previously we let T_n for $n \in \mathbb{N}$ denote the set of all functionals $\chi_n : B_n \to \{0, 1\}$, and by $W_{n,k}$ for $n, k \in \mathbb{N}$ with $k < n$ we denote the set of all operators $\phi_{n,k} : I^{n-k} \times B_n \to B_k$, where $I = \{1, \ldots, n\}$ is an index set.

Definition 2.6.2 (PROJECT) *The operator* $\mathrm{PROJECT}_{n,k} : T_n \to W_{n,k}$ *is defined for* $\chi_n \in T_n$, $f_n \in B_n$, *and* $\vec{\imath} \in I^{n-k}$ *by*

$$\mathrm{PROJECT}_{n,k}(\chi_n)(\vec{\imath}, f_n) = \mathrm{EACH}_{n-k,k}(\chi_n)(\mathrm{split}_{n,k}(\vec{\imath}, f_n)) \ ,$$

where the indices in $\vec{\imath}$ *are mutually distinct and have the functionality of pointing out the arguments of* f_n *to be eliminated by projection.*

To eliminate $n - k < n$ arguments of a Boolean function $f \in B_n$ by projection, we point out the indices of the arguments that we wish to eliminate using $\vec{\imath} \in I^{n-k}$ where $I = \{1, \ldots, n\}$ is an index set. We cannot eliminate the same argument more than once therefore the indices in $\vec{\imath}$ must be mutually distinct. The remaining arguments, that is, the axes in the image of f on which f is projected, are given by the indices in I which were not pointed out in $\vec{\imath}$. Let $\vec{\jmath} \in I^k$ denote these indices. The Boolean function $g \in B_k$ resulting from the projection, is independent of f, but the arguments of g will be valuated by the same propositional variables as the arguments of f pointed out by $\vec{\jmath}$. To find out whether one Boolean function implies another for any possible valuation, we have

Definition 2.6.3 (entail) *Let* $I = \{1, \ldots, n+m\}$ *be an index set with* $n, m \in \mathbb{N}$. *The functional*

$$\mathrm{entail}_{n,m} : I^n \times I^m \times B_n \times B_m \to \{0, 1\}$$

is defined for $\vec{\imath} \in I^n$, $\vec{\jmath} \in I^m$, $f_n \in B_n$, *and* $g_m \in B_m$ *by*

$$\mathrm{entail}_{n,m}(\vec{\imath}, \vec{\jmath}, f_n, g_m) = \wedge \ (\mathrm{fuse}((i_1, \ldots, i_n, j_1, \ldots, j_m), f_n \, \mathrm{OUTER}(\Rightarrow) \ g_m)) \ ,$$

where $\wedge$ *and* $\Rightarrow$ *denote the reduction transforms of the binary Boolean functions corresponding to the connectives denoted by the same symbols. If*

$$\mathrm{entail}_{n,m}(\vec{\imath}, \vec{\jmath}, f_n, g_m) = 1 \ ,$$

we write $f_n \models_{n,m} (\vec{\imath}, \vec{\jmath}) g_m$ *and say that* f_n entails g_m *in the given context.*

When a Boolean function $f \in B_n$ entails a Boolean function $g \in B_m$, it is said to be a *valid inference* to substitute f by g, but not conversely. Entailment can, hence, be referred to as the *correctness criterion* of a rule of inference. In Section 2.7, a rule of inference will be given a formal definition based on the functional entail. First, however, we will show that disjunctive projection is valid inference.

Theorem 2.6.4 *Let* $I = \{1, \ldots, n\}$ *be an index set, let* $A = A(p_1, \ldots, p_n)$ *be the formula describing a propositional rule set, and let* $f_A \in B_n$ *be the colligated form of a Boolean function represented by the rule set. Then for every index vector* $\vec{\imath} \in I^{n-k}$ *with* $k < n$ *in which all indices are mutually distinct, it holds that*

$$f_A \ \models_{n,k} \ ((1, \ldots, n), \vec{\jmath}) \mathrm{PROJECT}_{n,k}(\vee)(\vec{\imath}, f) \ ,$$

where $\vec{\jmath}$ *are the indices in* I *which are not in* $\vec{\imath}$.

Proof. Let $A = A(p_1, \ldots, p_n)$ be any formula describing a propositional rule set, and let $f_A \in B_n$ be the colligated Boolean function represented by it. Let $I = \{1, \ldots, n\}$ be the appropriate index set and pick an arbitrary $\vec{\imath} \in I^{n-k}$ with $k < n$ in which all indices are mutually distinct. Let $\vec{\jmath} \in I^k$ be given by the indices in I which are not in $\vec{\imath}$, then we have the following for an arbitrary valuation $a_i = v(p_i)$ with $i = 1, \ldots, n$:

$$\begin{aligned}
&\mathrm{entail}_{n,k}\left((1, \ldots, n), \vec{\jmath}, f_A, \mathrm{PROJECT}_{n,k}(\vee)(\vec{\imath}, f_A)\right) \\
&\quad = \bigwedge_{a_1, \ldots, a_n \in \{0,1\}} imp\left(f_A(a_1, \ldots, a_n), \mathrm{EACH}_{n-k,k}(\vee)(\mathrm{split}_{n,k}(\vec{\imath}, f_A))(\vec{a}_{\vec{\jmath}})\right) \\
&\quad = \bigwedge_{a_1, \ldots, a_n \in \{0,1\}} imp\left(f_A(a_1, \ldots, a_n), \mathrm{REDUCE}_{n-k}(vel)(\mathrm{split}_{n,k}(\vec{\imath}, f_A)(\vec{a}_{\vec{\jmath}}))\right) \\
&\quad = \bigwedge_{a_1, \ldots, a_n \in \{0,1\}} imp\left(f_A(a_1, \ldots, a_n), \bigvee_{b_1, \ldots, b_{n-k} \in \{0,1\}} \mathrm{split}(\vec{\imath}, f_A)(\vec{a}_{\vec{\jmath}})(b_1, \ldots, b_{n-k})\right) = 1 \ .
\end{aligned}$$

The last equality holds because for any value $f_A(a_1, \ldots, a_n) = 1$ there is a $\vec{b} = \vec{a_i}$ such that the same function value is a part of the disjunction given as the second argument of the implication imp. □

Disjunctive projection ($\mathrm{PROJECT}(\vee)$) corresponds, then, exactly to the syllogistic reasoning described introductorily. It is indeed interesting to note that we can draw a parallel between inference and projection in the image of a Boolean function. The depth of this observation will be explored in the next section after a simple example.

Example 2.6.5 Further elaborating on Example 2.5.4, we can find the relation between p_1 and p_2 according to the original rule set defined in Example 2.4.4. This is done by a disjunctive projection of the image of f_3 on the plane (in a logical coordinate-system) spanned by arguments one and three of f_3:

$$f_2 = \mathrm{PROJECT}_{3,2}(\vee)((2), f_3) \ .$$

We can then calculate that

$$\begin{aligned}
f_2(x_1, x_2) &= \mathrm{PROJECT}_{3,2}(\vee)((2), f_3)(x_1, x_2) \\
&= \mathrm{EACH}_{1,2}(\vee)(\mathrm{split}_{3,2}((2), f_3))(x_1, x_2) \\
&= \mathrm{REDUCE}_1(\vee)(\mathrm{split}_{3,2}((2), f_3)(x_1, x_2) \\
&= vel(f_3(x_1, 0, x_2), f_3(x_1, 1, x_2)) \\
&= vel(f_4(x_1, 0, 0, x_2), f_4(x_1, 1, 1, x_2)) \\
&= vel(et(imp(x_1, 0), imp(0, x_2)), et(imp(x_1, 1), imp(1, x_2))) \\
&= vel(et(non(x_1), 1), et(1, \mathit{aff}\,(x_2)) \\
&= vel(non(x_1), x_2)
\end{aligned}$$

Why we can conclude that $f_2 = imp$, proving the Aristotelian syllogism.

2.7 Rules of Inference

There seems to be no universal agreement upon a formal definition of inference. Nevertheless a correctness criterion for inference has been established by the concept of entailment. A rule of inference could then be described as a pair (A, C) where A is a propositional rule set and C is a propositional formula which A entails. Or it could be described, in terms of the Boolean functions which A and C represent, as an operator transforming a Boolean function such that the resulting Boolean function fulfils the correctness criterion by ways of the functional entail. It should, however, be noted that what we in the following refer to as trivial inference, some would not call inference at all since the inferred conclusion would be too obvious. The paradox of defining inference is well described by Jones [15]. The definition we adopt is very broad and the reader should feel free to confine our definition of a rule of inference for example by rejection of the rules of inference that we refer to as trivial. Only very few and simple corrections in the theory that follows would be necessary to accommodate a more restricted definition of inference.

Definition 2.7.1 (rule of inference) *A* rule of inference *is an operator $\chi : B_n \to B_m$ which, for at least one combination of $n \in \mathbb{N}$ and $m \in \mathbb{N}$, transforms at least one Boolean function $f \in B_n$ into a Boolean function in B_m such that $f \models_{n,m} (\vec{\imath}, \vec{\jmath})\chi(f)$ for some $\vec{\imath} \in \{1, \ldots, n+m\}^n$ and $\vec{\jmath} \in \{1, \ldots, n+m\}^m$.*

Let us take an example of how Definition 2.7.1 can be employed. In the following, we let **1** denote the constant function returning truth (1) for any argument, and we let **0** denote the constant function returning falsehood (0) for any argument.

Example 2.7.2 Suppose we have an operator $\psi : B_n \to B_n$ which transforms an arbitrary number of zeros (values of falsehood) in the image of its argument to ones (values of truth) in the image of the resulting transformed function. This rule can be defined by

$$\psi_g(f) = vel(f, g) \ ,$$

where $f, g \in B_n$. Observe that there are no inferences such that $f \models_{n,n} (\vec{\imath}, \vec{\imath})\psi(f)$, where $\vec{\imath} = (1, \ldots, n)$, which can not be described by this definition of ψ and

$$imp(f, vel(f, g)) = vel(non(f), vel(f, g)) = vel(vel(non(f), f), g) = vel(\mathbf{1}, g) = \mathbf{1}$$

ensures that there are no invalid inferences resulting from this rule of inference.

To be specific, the quite general rule of inference ψ can lead us to more well-known rules of inference. Suppose $g_1 = imp$ and $g_2 = cimp$. Now two well-known and very specific rules of inference appear when ψ_{g_1} and ψ_{g_2}, respectively, are applied to e.g. $f = bii$. We have

$$\psi_{g_1}(bii) = imp \quad \text{and} \quad \psi_{g_2}(bii) = cimp \ ,$$

or the corresponding representation in propositional calculus:

$$\frac{p \Leftrightarrow q}{p \Rightarrow q} \quad , \quad \frac{p \Leftrightarrow q}{p \Leftarrow q} \quad ,$$

where p and q are propositional variables and the expression above the line entails the expression below the line.

Note that the functional entail and, hence, rules of inference describe a relation between the values (images) of Boolean functions for some arguments, not a relation between Boolean functions in general. Therefore many different operators may describe the same rule of inference depending on the index vectors chosen for the entailment relation. To accommodate this construction, we introduce the concept of equivalent forms.

Definition 2.7.3 (equivalent forms) *Let $I = \{1, \ldots, k\}$ be an index set with $k = \max(n, m)$ and $n, m \in \mathbb{N}$. For $\vec{\imath} \in I^m$ let $A_1 = A_1(p_1, \ldots, p_n)$ and $A_2 = A_2(p_{i_1}, \ldots, p_{i_m})$ be propositional formulae, and let f_{A_1} and f_{A_2} be the Boolean functions which they represent. If A_1 and A_2 are logically equivalent, f_{A_1} and f_{A_2} are referred to as* equivalent forms.

For any two equivalent forms f_{A_1} and f_{A_2} there (trivially) exists a Boolean function $e_{\vec{\imath}} \in B_{m,n}$ such that

$$f_{A_1} = f_{A_2} \circ e_{\vec{\imath}} \ .$$

We let $\boldsymbol{e} \in B_{n,m}$, for any m, denote any Boolean function for which $f \circ \boldsymbol{e}$ is an equivalent form of $f \in B_n$.

Example 2.7.4 The idea of equivalent forms is merely to state that Boolean functions such as

$$f(a_1, a_2, a_3) = v((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_3))$$

and

$$g(a_2, a_3, a_2, a_1, a_4) = v((p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_3))$$

are equivalent forms, and we can use the symbol $\boldsymbol{e}$ to denote any equivalent form of f:

$$g(x_2, x_3, x_2, x_1, x_4) = (f \circ \boldsymbol{e})(x_2, x_3, x_2, x_1, x_4) = f(x_1, x_2, x_3) \ .$$

The notion of equivalent forms reveals a trivial rule of inference $\chi : B_n \to B_m$, which is valid for any $f \in B_n$. The rule is, of course, $\chi(f) = f \circ \boldsymbol{e}$ and we can observe that colligation and reduction of f to its reduced form $\mathrm{red}(f)$ (i.e. removal of all fictive arguments) is a trivial rule of inference which is contained in this formulation of χ. Another trivial rule of inference $\chi : B_n \to B_m$ which is valid for all $n, m \in \mathbb{N}$ and any $f \in B_n$, is $\chi(f) = \mathbf{1}$. We say that $\mathbf{1}$ is the *least restrictive form of f* and following the same line of thought $\mathrm{red}(f)$ is the *most restrictive form in B_n of $f \in B_n$*. The result of an inference is, however, often a Boolean function taking a different number of arguments. Therefore we introduce

Definition 2.7.5 (most restrictive form in B_m of $f \in B_n$) *Let $I = \{1, \ldots, n+m\}$ be an index set with $n, m \in \mathbb{N}$. For $\vec{\imath} \in I^n$ and $\vec{\jmath} \in I^m$ let $A_1 = A_1(p_{i_1}, \ldots, p_{i_n})$ and $A_2 = A_2(p_{j_1}, \ldots, p_{j_m})$ be propositional formulae, and let f_{A_1} and f_{A_2} be the Boolean functions which they represent. The* most restrictive form in B_m of $f_{A_1} \in B_n$ *is defined by the Boolean function $f_{A_2} \in B_m$ for which fewest arguments $\vec{a} \in \{0, 1\}^m$ exist such that $f_{A_2}(\vec{a}) = 1$, while it is true that $f_{A_1} \models_{n,m} (\vec{\imath}, \vec{\jmath}) f_{A_2}$.*

As a continuation of this definition let $f \in B_n$ and $g \in B_n$ denote two arbitrary Boolean functions. If there exists at least one more value of truth in the image of g than what exists in the image of f, that is, if there is at least one more argument $\vec{a} \in \{0,1\}^n$ such that $g(\vec{a}) = 1$ than there are arguments such that $f(\vec{a}) = 1$, then we say that g is *less restrictive* than f and conversely that f is *more restrictive* than g.

Pick two arbitrary Boolean functions $f, g \in B_n$. If f is more restrictive than g, then $f \models_{n,n} (\vec{\imath}, \vec{\imath})g$, where $\vec{\imath} = (1, \ldots, n)$, but conversely it is *not* the case that $g \models_{n,n} (\vec{\imath}, \vec{\imath})f$. Hence, we can always derive a less restrictive form from a more restrictive one, why if we have the most restrictive form of a relation, no new knowledge can be obtained from a less restrictive form. Hence, we are first of all interested in rules of inference which find the most restrictive form in B_m of $f_A \in B_n$, where A is the propositional formula representing f_A. Moreover it suffices to investigate rules of inference transforming the colligated form of a Boolean function since any propositional rule set in which n propositional variables appear, can be described by a colligated Boolean function $f \in B_n$.

Theorem 2.7.6 *Let $I = \{1, \ldots, n\}$ be an index set, and let $n, m \in \mathbb{N}$ be any two natural numbers. For all non-trivial rules of inference $\chi : B_n \to B_m$ taking a colligated Boolean function $f \in B_n$ as argument, there exists an $\vec{\imath} \in I^k$ with $0 < k \leq n$, such that*

$$\chi(f) = \mathrm{PROJECT}_{n,n-k}(\vee)(\vec{\imath}, f) \circ \boldsymbol{e}$$

is the most restrictive form in B_m of $f \in B_n$. If $k = 0$, there are only trivial rules of inference finding the most restrictive form in B_m of $f \in B_n$.

Proof. Let $\chi : B_n \to B_m$ be an arbitrary rule of inference defined for at least one argument in its colligated form. Suppose $f \in B_n$ is any one of the colligated Boolean functions which χ transforms into $\chi(f) \in B_m$ such that $f \models_{n,m} (\vec{\imath}, \vec{\jmath})\chi(f)$ for some $\vec{\imath} \in \{1, \ldots, n+m\}^n$ and $\vec{\jmath} \in \{1, \ldots, n+m\}^m$. The number of indices in $\vec{\imath}$ which are not in $\vec{\jmath}$ is denoted $k \in \mathbb{N}$. Note that $k \leq n$.

Let A_1 and A_2 be formulae representing f and $\chi(f)$ respectively in propositional calculus. Then the index vectors $\vec{\imath}$ and $\vec{\jmath}$ point out the propositional variables appearing in A_1 and A_2 which valuate the arguments of f and $\chi(f)$. Pick an arbitrary valuation v of the propositional variables appearing in A_1 and A_2, and let $\vec{a} \in \{0,1\}^n$ denote the corresponding argument of f and $\vec{b} \in \{0,1\}^m$ the corresponding argument of $\chi(f)$. From the definition of entailment (Def. 2.6.3), the options we have for χ are operators which ensure that if $f(\vec{a}) = 1$, then $\chi(f)(\vec{b}) = 1$, but if $f(\vec{a}) = 0$, then $\chi(f)(\vec{b})$ can return either true (1) or false (0).

For $k = 0$. All indices in $\vec{\imath}$ are also in $\vec{\jmath}$ why $m \geq n$. Since $m \geq n$, there exists an equivalent form in any set of Boolean functions B_m. An equivalent form $g \in B_m$ of f is also the most restrictive form in B_m of $f \in B_n$. This means that χ is given trivially as $\chi(f) = f \circ \boldsymbol{e}$ for all $n, m \in \mathbb{N}$ with $k = 0$. Hence, if $k = 0$, only trivial rules of inference exist finding the most restrictive form in B_m of f.

For $k = n$. No indices in $\vec{\imath}$ are also in $\vec{\jmath}$. If $f = \mathbf{0}$, then $\chi(f) = \mathbf{0}$ is the most restrictive form in B_m of f. If there is a single $\vec{a} \in \{0,1\}^n$ such that $f(\vec{a}) = 1$, it is necessary that $\chi(f) = \mathbf{1}$, since $\vec{a}$ could be the argument of f for any argument of $\chi(f)$. A functional which reduces f to 0 if $f = \mathbf{0}$ and 1 otherwise is $\mathrm{REDUCE}_n(vel)$ (cf. Def. 2.3.1). Since a Boolean value (0 or 1) could be regarded as a Boolean function taking no arguments, we have $\mathbf{0} = 0 \circ \boldsymbol{e}$ and $\mathbf{1} = 1 \circ \boldsymbol{e}$. But then

$$\chi(f) = \mathrm{REDUCE}_n(vel)(f) \circ \boldsymbol{e}$$

when $k = n$.

For $0 < k < n$. Let $\vec{\imath}' \in I^k$ be the indices in $\vec{\imath}$ which are *not* in $\vec{\jmath}$, and let $\vec{\jmath}' \in I^{m-n+k}$ be the indices in $\vec{\jmath}$ which are *not* in $\vec{\imath}$. Furthermore let $\vec{v} \in I^{n-k}$ be the indices which are *both* in $\vec{\imath}$ and in $\vec{\jmath}$. Using Def. 2.6.1 we have the following two equations

$$\begin{aligned} f(\vec{a}) &= \mathrm{split}_{n,n-k}(\vec{\imath}', f)(\vec{a}_{\vec{v}})(\vec{a}_{\vec{\imath}'}) \\ \chi(f)(\vec{b}) &= \mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{b}_{\vec{v}})(\vec{b}_{\vec{\jmath}'}) = \mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{a}_{\vec{v}})(\vec{b}_{\vec{\jmath}'}) \ . \end{aligned}$$

Since χ is a rule of inference, it must hold that $f \models_{n,m} (\vec{\imath}, \vec{\jmath})\chi(f)$. Hence, according to the definition of entailment (Def. 2.6.3), it follows that

$$imp(\mathrm{split}_{n,n-k}(\vec{\imath}', f)(\vec{a}_{\vec{v}})(\vec{a}_{\vec{\imath}'}), \mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{a}_{\vec{v}})(\vec{b}_{\vec{\jmath}'})) = 1 \ .$$

Recall that an arbitrary valuation was chosen, why this is true for all $\vec{a} \in \{0,1\}^n$ and $\vec{b} \in \{0,1\}^m$. But then it is also true that

$$\mathrm{split}_{n,n-k}(\vec{\imath}', f)(\vec{a}_{\vec{v}}) \models_{k,m-n+k} (\vec{\imath}', \vec{\jmath}')\mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{a}_{\vec{v}})$$

and since $h = \mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{a}_{\vec{v}}) \in B_{m-n+k}$ denotes a part of the image of $\chi(f)$, then for $\chi(f)$ to be the most restrictive form of $f \in B_n$, it is also necessary that h is the most restrictive form in B_{m-n+k} of $g = \mathrm{split}_{n,n-k}(\vec{\imath}', f)(\vec{a}_{\vec{v}}) \in B_k$.

Since no indices in $\vec{\imath}'$ are also indices in $\vec{\jmath}'$, the relation between g and h corresponds exactly to the case where $k = n$ described above. This means that if $g = \mathbf{0}$, then $h = \mathbf{0}$. Otherwise $h = \mathbf{1}$. This relation can be described perfectly by an operator $\psi : B_k \to B_{m-n+k}$ defined by $\psi(g) = h$.

At this point it should be observed that either

$$\chi(f)(\vec{b}) = \mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{a}_{\vec{v}})(\vec{b}_{\vec{\jmath}'}) = \psi(g)(\vec{b}_{\vec{\jmath}'}) = \mathbf{0}(\vec{b}_{\vec{\jmath}'})$$

or

$$\chi(f)(\vec{b}) = \mathrm{split}_{m,n-k}(\vec{\jmath}', \chi(f))(\vec{a}_{\vec{v}})(\vec{b}_{\vec{\jmath}'}) = \psi(g)(\vec{b}_{\vec{\jmath}'}) = \mathbf{1}(\vec{b}_{\vec{\jmath}'}) \ .$$

This means that the arguments pointed out by $\vec{\jmath}'$ have no influence whatsoever on the value returned by $\chi(f)$ (because we are finding the most restrictive form in B_m of $f \in B_n$). If we replace the operator ψ by a functional $\xi : B_k \to \{0,1\}$ defined by $\xi(g) = 0$ if $g = \mathbf{0}$, and $\xi(g) = 1$ otherwise, the fictive arguments can be represented by an equivalent form of $\xi \circ \mathrm{split}_{n,n-k}(\vec{\imath}', f)$ (cf. Def. 2.7.3). In other words,

$$\begin{aligned} \chi(f)(\vec{b}) &= \psi(g)(\vec{b}_{\vec{\jmath}'}) = \xi(g) = \xi(\mathrm{split}_{n,n-k}(\vec{\imath}', f)(\vec{a}_{\vec{v}})) = (\xi \circ \mathrm{split}_{n,n-k}(\vec{\imath}', f))(\vec{a}_{\vec{v}}) \\ &= (\xi \circ \mathrm{split}_{n,n-k}(\vec{\imath}', f))(b_{\vec{v}}) = ((\xi \circ \mathrm{split}_{n,n-k}(\vec{\imath}', f)) \circ \boldsymbol{e})(\vec{b}) \end{aligned}$$

which, using Definitions 2.3.2 and 2.6.2, can be rewritten as follows

$$\begin{aligned} \chi(f) &= (\xi \circ \mathrm{split}_{n,n-k}(\vec{\imath}', f)) \circ \boldsymbol{e} = \mathrm{EACH}_{k,n-k}(\xi)(\mathrm{split}_{n,n-k}(\vec{\imath}', f)) \circ \boldsymbol{e} \\ &= \mathrm{PROJECT}_{n,n-k}(\xi)(\vec{\imath}', f) \circ \boldsymbol{e} \ . \end{aligned}$$

From the definition of ξ we observe (in a similar manner as the case where $k = n$) that the functional $\mathrm{REDUCE}_k(vel) : B_k \to \{0,1\}$ is the exact equivalent of ξ for all $k \in \mathbb{N}$, why $\xi = \mathrm{REDUCE}_k(vel) = \vee$ (cf. Sec. 2.3). Finally if we return to the case where $k = n$, that is, where $\vec{a}_{\vec{\imath}'} = \vec{a}$, we have

$$\begin{aligned} \mathrm{REDUCE}_n(vel)(f) \circ \boldsymbol{e} &= \vee(\mathrm{split}_{n,n}(\vec{\imath}, f)()) \circ \boldsymbol{e} = (\vee \circ \mathrm{split}_{n,n}(\vec{\imath}, f))() \circ \boldsymbol{e} \\ &= \mathrm{PROJECT}_{n,0}(\vee)(\vec{\imath}, f)() \circ \boldsymbol{e} = \mathrm{PROJECT}_{n,0}(\vee)(\vec{\imath}, f) \circ \boldsymbol{e} \ . \end{aligned}$$

Therefore, since the rule of inference $\chi : B_n \to B_m$ was chosen arbitrarily, and since the valuation v was chosen arbitrarily, we can conclude that for all non-trivial rules of inference $\chi : B_n \to B_m$ taking a colligated Boolean function $f \in B_n$ as argument, there exists an $\vec{\imath}' \in I^k$ with $0 < k \leq n$, such that

$$\chi(f) = \mathrm{PROJECT}_{n,n-k}(\vee)(\vec{\imath}', f) \circ \boldsymbol{e}$$

finds the most restrictive form in B_m of $f \in B_n$. Proving exactly what was required. □

A corollary follows immediately from Theorem 2.7.6, Example 2.7.2, and the notion of a colligated Boolean function (Def. 2.5.3).

Corollary 2.7.7 *Let $I = \{1, \ldots, n\}$ be an index set, and let $n, n', m, k \in \mathbb{N}$ be natural numbers such that $k \leq n \leq n'$. For every rule of inference $\chi' : B_{n'} \to B_m$ there is an equivalent rule of inference $\chi : B_n \to B_m$ transforming the colligated equivalent of f', namely $f \in B_n$, into the same Boolean function $\chi(f) = \chi'(f') \in B_m$. For the equivalent rule of inference χ, there exist, if $k > 0$, a $g \in B_{n-k}$ and an $\vec{\imath} \in I^k$ such that*

$$\chi_g(f) = (g \vee \mathrm{PROJECT}(\vee)(\vec{\imath}, f)) \circ \boldsymbol{e} \ .$$

For $k = 0$ there exist a $g \in B_n$ such that

$$\chi_g(f) = (g \vee f) \circ \boldsymbol{e} \ .$$

This means that orthogonal projection and union of Boolean functions in a many-dimensional logical coordinate system, is all we need for any kind of inference in propositional logic. While it follows from Theorem 2.7.6 that all the most restrictive forms resulting from inference can be found through projection, we emphasize that Corollary 2.7.7 provides a formula from which any rule of inference for propositional logic can be obtained. Still the most restrictive forms are the most interesting forms that we can infer. The reason being, as mentioned before, that we can derive a less restrictive form from a more restrictive one, but not conversely. Therefore the rules finding the most restrictive forms are sometimes the only rules which are accepted as true rules of inference. This point of view is reflected in the early analogies between logic and algebra.

De Morgan writes [21, p. 27]: "Speaking instrumentally, what is called *elimination* in algebra is what is called *inference* in logic."[1] And since elimination in algebra can be accomplished through orthogonal projection of a surface on the space spanned by a few axes, this indicates that De Morgan had thoughts about inference similar to what we arrive at in Theorem 2.7.6. He even states that "we can compare the forms of logic in reasoning with the laws of linear perspective in painting" [21, pp. 26–27], thereby coming even closer to the analogy between projection and inference (except for the fact that we employ orthogonal projection rather than the perspective projection used by a painter trying realistically to reproduce a three-dimensional scene).

Since Boole was also working on an analogy between logic and algebra, he was investigating the relation between inference and elimination. In fact one of the key points in his celebrated Laws of Thought is to address the question "Whether deductive reasoning can with propriety be regarded as consisting only of elimination" [2, pp. 239–240], and he writes subsequently: "I reply, that reasoning cannot, except by an arbitrary restriction of its meaning, be confined to the process of elimination". In support of Boole's conclusion, Peirce remarked in a footnote that "De Morgan ("On the Syllogism," No. II., 1850, p. 84) goes too far [...] if he means, as he seems to do, that all inference is elimination" [25, §184n]. And this is exactly what we have also discovered. The arbitrary function g in Corollary 2.7.7 has exactly the purpose of including all the rules of inference which can not be described by projection/elmination, namely those resulting in a less restrictive form. Boole and Peirce may have rejected the idea of all inference as elimination because of a similar observation.

Boole's analogy between logic and algebra was founded in the idea of two-valued polynomial functions to represent propositional rule sets, De Morgan's approach, on the other hand, was oriented towards systems of linear equations or inequalities to represent the same thing. In the tradition of De Morgan it was discovered in 1991 by Hooker [12] that everything which can be inferred from a rule set about a restricted set of propositions, can be found through logical projection. A result which is very similar to our Theorem 2.7.6. The context and definitions prior to Hooker's proof is, however, completely different from and not as general as ours. The reason being that any propositional formula must be rewritten as a clause in Hooker's treatment whereas we impose no restrictions on the representation of Boolean functions represented by a propositional formula. As De Morgan, Hooker also does not mention rules leading to less restrictive forms.

Having now described how inference can be drawn on a rule set by projection, it may be that we have a rule set specifying the relation between n propositional variables. Suppose we want to assert truth or falsehood to a number $k < n$ of these propositions and draw a conclusion on the relation between the remaining propositions in the rule set. To do inference by projection, it would be necessary to include the assertions in the rule set. In the following section we describe a simpler option for drawing inference in this special case where a number of propositions are simply asserted.

2.8 Deduction

The nesting of a Boolean function employed in projection, is useful not only for syllogistic reasoning, but also for the form of deductive reasoning described in the Stoic modi. This is a form of inference resulting from external influences such as the consequence of some propositional variable being asserted (or valuated as) true or false.

Knowing the current value of one or several propositional variables appearing in a formula A, makes us able to pick a subspace in the image of a Boolean function f_A represented by A. The picking of a subspace involves no calculations and is therefore much more efficient than inference by projection. We have

Definition 2.8.1 (deduce) *Let $I = \{1, \ldots, n\}$ be an index set, and let $n, k \in \mathbb{N}$ be natural numbers with $k < n$. Then the operator* $\mathrm{deduce} : \{0,1\}^k \times I^{n-k} \times B_n \to B_{n-k}$ *is defined for $\vec{a} \in \{0,1\}^k$, $\vec{\imath} \in I^{n-k}$, and $f_n \in B_n$*

[1] Italicizations are original.

by

$$\mathrm{deduce}_{n,n-k}(\vec{a},\vec{\imath},f_n) = \mathrm{split}_{n,k}(\vec{\imath},f_n)(\vec{a}) \ ,$$

where the indices in $\vec{\imath}$ must be mutually distinct. Note that $\vec{\imath}$ points out the arguments of f_n that have not been asserted (unknowns).

Theorem 2.8.2 *Let $I = \{1,\ldots,n\}$ be an index set with $n \in \mathbb{N}$. Let $A = A(p_1,\ldots,p_n)$ be a formula describing a propositional rule set, and let $f_A \in B_n$ be the Boolean function represented by A. For any $\vec{\imath} \in I^m$ let $v : \{p_{i_1},\ldots,p_{i_m}\} \to \{0,1\}$ be a partial valuation for A such that $a_k = v(p_{i_k})$ for $k = 1,\ldots,m$ according to an assertion of the propositional variables $p_{i_1},\ldots,p_{i_k}$. It then holds that*

$$f \models_{n,n-m} ((1,\ldots,n),\vec{\jmath})\mathrm{deduce}_{n,n-m}(\vec{a},\vec{\jmath},f) \ ,$$

where $\vec{\jmath} \in I^{n-m}$ is given by the indices in I which are not in $\vec{\imath}$ ordered such that $j_k < j_{k+1}$ for $k = 1,\ldots,n-m-1$.

Proof. Pick an arbitrary propositional rule set and let $f_A \in B_n$ be the Boolean function represented by the formula $A = A(p_1,\ldots,p_n)$ describing the rule set. Suppose an external influence asserts any partial valuation v for A such that $a_k = v(p_{i_k})$, where $k = 1,\ldots,m$ and $\vec{\imath} \in I^m$ for $I = \{1,\ldots,n\}$. Let furthermore $\vec{\jmath} \in I^{n-m}$ be the indices in I which are not in $\vec{\imath}$ ordered such that $j_k < j_{k+1}$ for $k = 1,\ldots,n-m-1$. Then

$$\begin{aligned}&\mathrm{entail}_{n,n-m}((1,\ldots,n),\vec{\jmath},f_A,\mathrm{deduce}_{n,n-m}(\vec{a},\vec{\jmath},f_A))\\ &\quad = \bigwedge_{b_1,\ldots,b_n\in\{0,1\}} imp(f_A(b_1,\ldots,b_n),\mathrm{split}(\vec{\jmath},f_A)(\vec{a})(b_{j_1},\ldots,b_{j_{n-m}})) \ .\end{aligned}$$

This holds true only as long as $b_{i_k} = a_k = v(p_{i_k})$ for $k = 1,\ldots,m$ which is the case as long as the propositional variables $p_{i_1},\ldots,p_{i_k}$ are asserted. □

Example 2.8.3 Consider the simple propositional rule set, or formula,

$$p \Rightarrow q \ .$$

This rule set, of course, represents the Boolean function $f_2 = imp$.

Suppose we have an external influence asserting that p is *true*. Then we have deductively that

$$f_1 = \mathrm{deduce}_{2,1}((1),(2),f_2) = \mathrm{split}_{2,1}((2),f_2)(1)$$

and

$$\begin{aligned}f_1(0) &= \mathrm{split}_{2,1}((2),f_2)(1)(0) = f_2(1,0) = imp(1,0) = 0\\ f_1(1) &= \mathrm{split}_{2,1}((2),f_2)(1)(1) = f_2(1,1) = imp(1,1) = 1 \ ,\end{aligned}$$

why we can conclude that for the rule set to be fulfilled, the consequence of p being *true* is that q is *true* (cf. Table 1). This proves modus ponens.

To conclude on the theory that has been presented, we emphasize that all kinds of deductive inference on arbitrary rule sets can be performed by a disjunctive projection in a logical coordinate system (and disjunction of any Boolean function as described in Corollary 2.7.7, if we need a less restrictive form of a conclusion). Moreover we can perform the simpler picking of a subspace described in this section, to draw inference on simple assertions of propositions. We find it advantageous to think of assertions as influences external to the rule set rather than additional rules which should be added to the set. The reason is that many systems can be described by a static rule set and for that we can compute the image of the Boolean function represented by the rule set, in advance. Then the efficient picking of a subspace can quickly narrow down the part of the image which we need to consider to find the logical consequences of dynamically changing input to the system.

It should be observed that both ways to draw inference can be done mechanically by an implementation of EACH and split (and fuse if functions are not colligated in advance). We can even find the image of Boolean function representing arbitrary propositional rule sets using OUTER and fuse which is more efficient than testing every possible valuation for the rule set. In the following section we investigate how the operators can be applied to different representations of Boolean functions.

3 Representations of Boolean Functions

We will, shortly, give examples of how the operators presented in the previous section can be applied to both polynomial, table, and graph representations of Boolean functions.

3.1 Polynomial Representations

Boolean functions were originally presented by Boole through a polynomial development formula where a logical variable x can attain only the truth-values 0 and 1, $(1-x)$ denotes the negation of x, logical multiplication corresponds to conjunction, and logical addition corresponds to disjunction. A description of Boole's development process has been given by Franksen [9]. Boole's polynomial representation has lead to the notion of Boolean algebra (see eg. [29]) and the polynomial representation of Boolean functions is still used extensively.

In a Boolean algebra different *normal forms* can provide a basis for the space of Boolean functions expressed as polynomials. The disjunctive normal form is the one most commonly chosen and it expresses the polynomials in the form of an $\vee$-sum of $\wedge$-product terms. Another interesting approach is to use exclusive-or ($\oplus$) instead of disjunction for logical addition. The exclusive-or normal form allows for a true vector space of Boolean functions, since the operations $\wedge$ and $\oplus$ are, in fact, the modulo-two product and sum which comprise a field, see [6]. This is not the case if the disjunctive normal form is employed.

Regardless of the choice of basis, the presented operators can easily be applied to any polynomial representation of Boolean functions.

Example 3.1.1 Again consider the rule set in Example 2.4.4. Giving the Boolean functions corresponding to the rules a polynomial representation results in

$$\begin{aligned} r_1(a_1,a_2) = &\quad v(p_1 \Rightarrow p_2) &&= a_1a_2 + (1-a_1)a_2 + (1-a_1)(1-a_2) \\ r_2(a_2,a_3) = &\quad v(p_2 \Rightarrow p_3) &&= a_2a_3 + (1-a_2)a_3 + (1-a_2)(1-a_3) \ , \end{aligned}$$

where $v(A)$ is the value of the formula A by the valuation $v : \{p_1,p_2,p_3\} \to \{0,1\}$ such that $a_i = v(p_i)$, $i = 1,2,3$. Multiplication corresponds to conjunction and addition corresponds to disjunction. The polynomial representation written after the second equality is the disjunctive normal form of the rule. Now everything works as in the previous examples. First we can construct a Boolean function $f_3 \in B_3$ corresponding to the rule set

$$\begin{aligned} f_4 &= r_1 \, \mathrm{OUTER}_{2,2}(\cdot) \, r_2 \\ f_3 &= \mathrm{fuse}_{4,3}((1,2,2,3), f_4) \ . \end{aligned}$$

Again reasoning is easily captured. The relation between p_1 and p_3 is given as the Boolean function f_2 obtained by

$$f_2 = \mathrm{PROJECT}_{3,2}(+)((2), f_3) \ .$$

Observe that nothing done after statement of the rules on disjunctive normal form, has demanded calculation. All we have done is simple substitution using the operators and movement of parentheses. In this representation of Boolean functions, the advantage of the operators lies in the *delay of calculations*. Using the operators we can keep track of the functions that are currently requested (such as f_2), but only when some specific result is needed a calculation is done. For example if we decide that the representation of f_2 on disjunctive normal form is the requested output, we could arrive at the result in the following way for a valuation $v : \{p_1,p_3\} \to \{0,1\}$ such

that $a_1 = v(p_1)$ and $a_2 = v(p_3)$:

$$\begin{aligned}
f_2(a_1, a_2) &= \mathrm{PROJECT}_{3,2}(+)((2), f_3)(a_1, a_2) \\
&= \mathrm{EACH}_{1,2}(+)(\mathrm{split}_{3,2}((2), f_3))(a_1, a_2) \\
&= \mathrm{REDUCE}_1(+)(\mathrm{split}_{3,2}((2), f_3)(a_1, a_2)) \\
&= \sum_{b\in\{0,1\}} f_3(a_1, b, a_2) \\
&= \sum_{b\in\{0,1\}} \mathrm{fuse}_{4,3}((1,2,2,3), f_4)(a_1, b, a_2) \\
&= \sum_{b\in\{0,1\}} \mathrm{OUTER}_{2,2}(\cdot)(r_1, r_2)(a_1, b, b, a_2) \\
&= \sum_{b\in\{0,1\}} r_1(a_1, b) r_2(b, a_2) \\
&= (1 - a_1)(a_2 + (1 - a_2)) + (a_1 + (1 - a_1))a_2 \\
&= a_1 a_2 + (1 - a_1)a_2 + (1 - a_1)(1 - a_2) \ .
\end{aligned}$$

Delay of operations is important if we want to do propositional reasoning in a dynamic environment such as a multi-agent system where the desired conclusion of each agent may change continuously according to real-time sensory input.

3.2 Table Representations

Truth tables are the most well-known table representation of Boolean functions. There are, however, many alternatives. Karnaugh maps [16] comprise an attempt to have a compact table representation. In the following we will, again, go over the Aristotelian syllogism and show the process of applying the operators in their original settings, namely in Franksen's array-based logic.

The arrays are ordered, orthogonal, and many-dimensional. They must also allow for one level of nesting to enable the concept of nested Boolean functions. When drawing the image of a real-valued function, the axes are distinguished as an arrow marked with the variable that it represents. In array-based logic no axes are drawn explicitly. Rather the structure of an array indicates which variable each axis corresponds to. The last axis of an array, corresponding to the valuation given as the last argument to the Boolean function, is always innermost and horizontal. Moving backwards through the list of arguments, the corresponding axes alternate between vertical and horizontal directions. This way of constructing arrays is due to Trenchard More [19, 20].

Example 3.2.1 For one last time consider the rule set in Example 2.4.4. Describing *aff* by an array we have:

$$\mathit{aff} = \begin{array}{cc} 0 & 1 \end{array} \ .$$

An array representing the Boolean function $f_3 \in B_4$ corresponding to the rule set is then found in the following way:

$$r_1 = r_2 = \mathit{aff}\ \mathrm{OUTER}_{1,1}(\Rightarrow)\ \mathit{aff} = \begin{array}{cc} 0 & 1 \end{array}\ \mathrm{OUTER}_{1,1}(\Rightarrow)\ \begin{array}{cc} 0 & 1 \end{array} = \begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array}$$

$$f_4 = r_1\ \mathrm{OUTER}_{2,2}(\wedge)\ r_2 = \begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array}\ \mathrm{OUTER}_{2,2}(\wedge)\ \begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} = \begin{array}{cc|cc} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array}$$

$$f_3 = \mathrm{fuse}((1,2,2,3), f_4) = \begin{array}{cc|cc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{array} \ .$$

If the relation f_2 between the variables p_1 and p_3 is desired, the projection is accomplished as follows:

$$\begin{aligned} f_2 &= \mathrm{PROJECT}_{3,2}(\vee)((2), f_3) = \mathrm{EACH}_{1,2}(\vee)\left(\mathrm{split}_{3,2}\left((2),\ \begin{matrix}1\,1 & 0\,0\\ 0\,1 & 0\,1\end{matrix}\right)\right) \\ &= \mathrm{EACH}_{1,2}(\vee)\left(\begin{array}{c|c}1\,0 & 1\,1\\ \hline 0\,0 & 0\,1\end{array}\right) = \begin{matrix}1\,1\\ 0\,1\end{matrix} \quad . \end{aligned}$$

The arrays presented here could, of course, be given a more compact form, and it is worth noticing that reasoning on arbitrary rule sets is easily automated through implementation of the operators. At least this is the case using table representations, and it has traditionally been done in array-based logic, see eg. [18].

An interpreted development language called Q'Nial (*Q*ueen's University *N*ested *i*nter*a*ctive *l*anguage) was originally proposed in [14] for the purpose of testing array theoretic concepts. If the reader feels a need to test the operators in an array theoretic setting, we recommend Q'Nial[2] where an implementation of EACH, OUTER, split, and fuse is available for operation on multi-dimensional nested arrays. The index argument of fuse is different in Q'Nial as compared to our definition, but the same functionality can be obtained with either definition. Another option for testing the presented operators is APL[3] which also has a nested array data structure readily available.

3.3 Graph Representations

There are many ways to represent Boolean functions as a graph. Most of them are based on a normal form just as the polynomial representations are. The most commonly known graph representation of a Boolean function is a Binary Decision Diagram (BDDs) which is based on the if-then-else normal form (see eg. [1]).

Constructing the graph representing a Boolean function is not necessarily straight forward. Suppose we need to merge two Boolean functions and draw inference on them in a time constrained environment where we cannot afford to construct a new merged graph. In that case OUTER and fuse can be used to merge the two graphs artificially, see Figure 1. There is also the possibility that the operators can be given an efficient implementation for BDDs and teach us new things about efficient construction of graph structures representing Boolean functions.

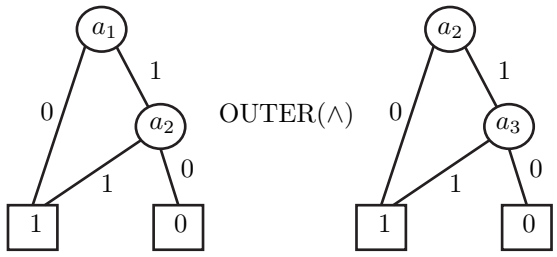

Fig. 1 The operators OUTER and fuse can be used to merge two graphs artificially. Connection of the two Binary Decision Diagrams representing Rule 1 and Rule 2 (from Example 2.4.4), is illustrated here.

Assertion of propositions is easily captured in a BDD through restriction of the graph. Syllogistic reasoning is, however, not obvious. The PROJECT operator may be able to help in this context, see Figure 2.

4 Discussion and Conclusion

Operators working on the images of Boolean functions, sometimes with a direct analogy to geometrical operators in cartesian spaces (compare $\mathrm{PROJECT}(\vee)$ and orthogonal projection), have been presented in an abstract form. Their purpose is to clarify each step of propositional reasoning regardless of the underlying representation of the Boolean functions.

In particular we have shown that the image of a Boolean function, say f, represented by an arbitrary propositional rule set, can be found using outer products and the operation of setting indices equal (fuse), the latter

[2] Q'Nial is available at http://www.nial.com/.

[3] For example dyalog APL http://www.dyalog.com/.

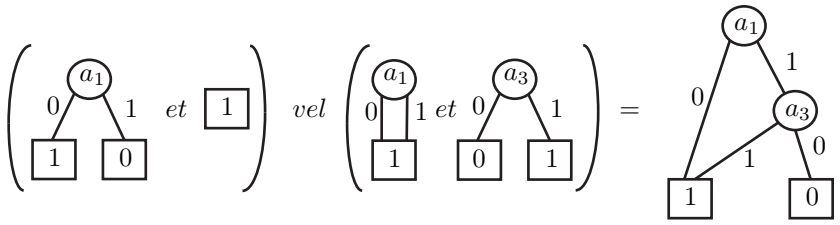

Fig. 2 Here $\mathrm{PROJECT}(\vee)$ is invoked on the a_2 argument of the graph shown in Figure 1 (the projection returns the disjunction between the graph where $v(p_2) = a_2 = 0$ and the graph where $v(p_2) = a_2 = 1$). Of course $\mathrm{PROJECT}$ can also be used with a Binary Decision Diagram that has not been connected using OUTER.

of which geometrically corresponds to the picking of diagonal hyperplanes in a logical coordinate system. If employed in the right way, these two operators can significantly limit the number of calculations needed for determining the image of f. At least this is the case if we compare to finding the value of the Boolean function by testing every possible valuation for the rule set. This technique is not new, but we have redefined the necessary operators in a form which is independent of the representation of the Boolean function describing the rule set.

In addition and, perhaps, more importantly, we have provided a formula from which any rule of inference can be derived. The formula shows that any rule of inference is captured by disjunctive projection in and union of Boolean function images in a logical coordinate system. This also means that disjunctive projection (and union to find less interesting consequences) is all we need to find any possible logical consequence which results if we want to fulfill any given propositional rule set.

A few examples of application of the operators to different representations of a propositional rule set have been described. This is an area in which much work can still be done. The impact of the operators have hitherto only been thoroughly investigated in the context of an array-based representation.

As mentioned previously, it may be an advantage to redefine the operators using the *Boolean tensors* described by Mautner [17]. But then it is necessary to introduce a considerable amount of algebra.

Finally there are many generalizations of this theory which come easily. For example there is no difficulty in rewriting the operators to be defined on the more general *Boolean-valued* functions represented by polyvalent logic. This merely has the result that the axes in the logical coordinate systems grow longer. The described operators (in their array theoretic form) have previously been employed in many different contexts, sometimes for propositional logic, but also in slightly more general settings. Examples of application areas are logic control of electronic apparatus [18], railway interlocking systems [27], automated approximate reasoning and fuzzy logic control [13], power system control [22], automated real-time decision systems for e-commerce [3, 5, 4], and agents in real-time environments [11]. This is, however, the first paper proving formally that the operators perform correct inference on any propositional rule set and that they can capture any possible kind of inference in propositional logic. This makes us confident that the operators are useful in many contexts.

Acknowledgement

Thanks to Vagn Lundsgaard Hansen for a few insightful discussions concerning the presented theory and to Jørgen Fischer Nilsson for a helpful comments. Last, but certainly not least, thanks to our anonymous reviewer for correction of untraditional formulations and for directions improving Definitions 2.3.1 and 2.6.3.

References

[1] H. R. Andersen. An introduction to Binary Decision Diagrams. Lecture notes for 49285 Advanced Algorithms E97. Department of Information Technology, Technical University of Denmark, October 1997.

[2] G. Boole. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities*. Dover Publications, Inc., New York, 1958. The first American printing of the work originally published by Macmillan in 1854.

[3] R. Davidrajuh. *Automating Supplier Selection Procedures*. PhD thesis, Narvik Institute of Technology, 2000.

[4] R. Davidrajuh. Modeling and implementation of supplier selection procedures for e-commerce initiatives. *Industrial Management and Data Systems*, 103(1):28–39, 2003.

[5] R. Davidrajuh and B. Hussein. Modeling logic systems with structured array-based logic. *Modeling, Identification and Control*, 24(1):27–36, 2003.

[6] H. Fleisher, M. Travel, and J. Yeager. Exclusive-OR representation of Boolean functions. *IBM Journal of Research and Development*, 27(4):412–416, 1983.

[7] O. I. Franksen. Group representation of finite polyvalent logic: A case study using APL notation. In A. Niemi, editor, *A Link between Science and Applications of Automatic Control, IFAC VII, World Congress 1978*, pages 875–887, Oxford, 1979. Pergamon Press.

[8] O. I. Franksen. Invariance under nesting - an aspect of array-based logic with relation to Grassmann and Peirce. In G. Schubring, editor, *Hermann Günther Graßmann (1809-1877): Visionary Mathematician, Scientist and Neohumanist Scholar*, pages 303–335, Dordrecht, 1996. Kluwer Academic Publishers.

[9] O. I. Franksen. Boole's development process revisited: From an array-theoretic viewpoint. *Acta historica Leopoldina*, 27:175–188, 1997.

[10] O. I. Franksen and P. Falster. Colligation or, the logical inference of interconnection. *Mathematics and Computers in Simulation*, 52(1):1–9, March 2000.

[11] J. R. Frisvad, P. Falster, G. L. Møller, and N. J. Christensen. Knowledge exchange between agents in real-time environments. In *Proc. of the International Conference on Computer Animation and Social Agents (CASA 2005)*, pages 127–132. The Hong Kong Polytechnic University, October 2005.

[12] J. N. Hooker. Logical inference and polyhedral projection. *Lecture Notes in Computer Science*, 626:184–200, 1991. Proceedings of the 5th Workshop on Computer Science Logic.

[13] J. Jantzen. Array approach to fuzzy logic. *Fuzzy Sets and Systems*, 70(2–3), 1995.

[14] M. A. Jenkins. A development system for testing array theory concepts. *ACM SIGAPL APL Quote Quad*, 12(1):152–159, September 1981.

[15] E. E. C. Jones. The paradox of logical inference. *Mind*, 7(26):205–218, April 1898.

[16] M. Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the AIEE*, 72(9):593–599, 1953.

[17] F. I. Mautner. An extension of klein's erlanger program: Logic as invariant-theory. *American Journal of Mathematics*, 68(3):345–384, July 1946.

[18] G. L. Møller. *On the Technology of Array-Based Logic*. PhD thesis, Electrical Power Engineering Department, Technical University of Denmark, 1995. Available at http://www.arraytechnology.com/.

[19] T. More, Jr. Axioms and theorems for a theory of arrays. *IBM Journal of Research and Development*, 17(2):135–175, 1973.

[20] T. More, Jr. The nested rectangular array as a model of data. In *Proceedings of the International Conference on APL: Part 1*, pages 55–73, 1979.

[21] A. D. Morgan. On the syllogism: II. *Transactions of the Cambridge Philosophical Society*, IX:79–127, 1850. Reprinted in, Peter Heath editor, *On the Syllogism and Other Logical Writings* by Augustus De Morgan, Routledge & Kegan Paul Limited, 1966.

[22] C. Nesgaard. An array-based study of increased system lifetime probability. In *Proceedings of IEEE Workshop on Computers in Power Electronics (COMPEL 2002)*, pages 82–86, June 2002.

[23] A. Pedersen. *Digraph Representation in Array-Based Logic: With Special Emphasis on the Mathematical Foundation of Production Models*. PhD thesis, Electrical Power Engineering Department, Technical University of Denmark, September 1992.

[24] C. S. Peirce. Grand logic (1893). In C. Hartshorne and P. Weiss, editors, *Collected Papers of Charles Sanders Peirce*, volume II, Book III. Harvard University Press, second printing, 1960.

[25] C. S. Peirce. On the algebra of logic, *American Journal of Mathematics*, vol. 3, pp. 15–57 (1880). In C. Hartshorne and P. Weiss, editors, *Collected Papers of Charles Sanders Peirce*, volume III, Paper IV. Harvard University Press, second printing, 1960.

[26] E. L. Post. Introduction to a general theory of elementary propositions. *Americal Journal of Mathematics*, 43:163–185, 1921. Reprinted in, Jean van Heijenoort editor, *From Frege to Gödel: A Source Book in Mathematical Logic 1879–1931*, pp. 264–283. Harvard University Press, Cambridge, 1967.

[27] C. Strunge. Applying array-based logic to substation control for switch interlocking. *IEEE Transactions on Power Delivery*, 14(3):879–883, July 1999.

[28] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons Ltd, and B. G. Teubner, Stuttgart, 1987.

[29] J. E. Whitesitt. *Boolean Algebra and Its Applications*. Dover Publications, Inc., New York, 1995. First published by the Addison-Wesley Publishing Company, Reading, Massachusetts, in 1961.